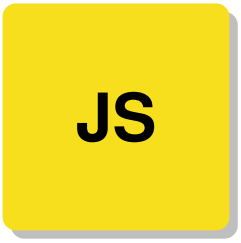
# FULL STACK

## DEVELOPMENT




JavaScript, Bootstrap, ReactJS & MongoDB

Essential Concepts for Modern Web Development

**Author: Rama Bhadra Rao Maddu**

**Version 1.0 — September 1, 2025**

## Copyright Notice



**Publisher:** Self Published
**Author:** Rama Bhadra Rao Maddu
**First Edition:** 2024

## Book Objectives

The main objective of this course is to provide comprehensive understanding on:

- Essential JavaScript concepts for web development
- Bootstrap framework for responsive design
- ReactJS for building modern user interfaces
- MongoDB for NoSQL database management
- Full-stack application development workflow

This book bridges the gap between frontend and backend development, enabling readers to build complete web applications from scratch.

# Contents

# 1. BASIC JAVASCRIPT

## Introduction to JavaScript

JavaScript is a high-level, interpreted programming language that has become one of the core technologies of the World Wide Web. Created by Brendan Eich in 1995 in just 10 days at Netscape Communications, JavaScript has evolved from a simple scripting language to power complex web applications, server-side development through Node.js (created by Ryan Dahl in 2009), and even mobile applications.

**Key Characteristics of JavaScript:**

- **Dynamic Typing:** Variables can hold values of any type without declaring the type
- **Interpreted:** No compilation step required; code is executed directly
- **Prototype-based:** Objects can inherit directly from other objects
- **First-class Functions:** Functions are treated as values
- **Event-driven:** Responds to user interactions and system events
- **Multi-paradigm:** Supports procedural, object-oriented, and functional programming

**JavaScript Engines:**

JavaScript code is executed by JavaScript engines. Each browser uses a different engine:

- **V8:** Used by Google Chrome and Node.js (developed by Google)
- **SpiderMonkey:** Used by Mozilla Firefox
- **JavaScriptCore:** Used by Safari (Apple)
- **Chakra:** Used by Internet Explorer and Legacy Microsoft Edge

**Client-side vs Server-side JavaScript:**

- **Client-side (Browser):**
  - Runs in web browsers
  - Manipulates DOM
  - Handles user interactions
  - Makes AJAX requests
  - Has access to browser APIs
  - Restricted file system access for security
- **Server-side (Node.js):**
  - Runs on servers
  - Full file system access
  - Can connect to databases
  - Creates HTTP servers
  - No DOM access
  - Can use npm packages

## JavaScript Instructions and Statements

A JavaScript program is a list of programming statements or instructions. Each statement performs a specific action and ends with a semicolon (though semicolons are optional in many cases due to automatic semicolon insertion).

Listing 1.1: **Basic JavaScript Statements**

```
// A simple statement
console.log("Hello, World!");

// Multiple statements
let message = "Learning JavaScript";
console.log(message);

// Statements can span multiple lines
let longMessage = "This is a very long message that " +
                  "spans multiple lines for readability";

// Compound statement (block)
{
    let x = 5;
    let y = 10;
    console.log(x + y);
}

// Statement types examples
let declaration = "This is a declaration statement";
5 + 3; // Expression statement
if (true) { /* Control flow statement */ }
function example() { return; } // Jump statement
```

**Statement Types:**

- **Expression statements:** Evaluate to a value
- **Declaration statements:** Declare variables or functions
- **Control flow statements:** Control program execution
- **Jump statements:** Transfer control

## Comments in JavaScript

Comments are essential for code documentation and are ignored by the JavaScript engine during execution.

Listing 1.2: **Types of Comments in JavaScript**

```
// Single-line comment
// This explains what the next line does
let userName = "John";
```

```
4
5  /*
6     Multi-line comment
7     This is useful for longer explanations
8     or temporarily disabling code blocks
9  */
10
11 let age = 25; // Inline comment
12
13 /**
14  * JSDoc comment for documentation
15  * @param {string} name - The user's name
16  * @param {number} age - The user's age
17  * @returns {string} A greeting message
18  */
19 function createGreeting(name, age) {
20     return `Hello ${name}, you are ${age} years old`;
21 }
22
23 // TODO: Add validation
24 // FIXME: Handle edge cases
25 // NOTE: This is deprecated
```

## Variables in JavaScript

Variables are containers for storing data values. JavaScript has evolved from using only `var` to modern `let` and `const` declarations.

### Variable Declaration Keywords

Listing 1.3: **Variable Declaration Methods - var, let, const**

```
1  // var - Function-scoped (ES5 and earlier)
2  var oldVariable = "I'm function-scoped";
3  var canBeRedeclared = "first";
4  var canBeRedeclared = "second"; // Allowed
5
6  // let - Block-scoped (ES6+)
7  let modernVariable = "I'm block-scoped";
8  let cannotBeRedeclared = "only once";
9  // let cannotBeRedeclared = "error"; // SyntaxError
10
11 // const - Block-scoped, cannot be reassigned (ES6+)
12 const constantVariable = "I cannot be reassigned";
13 // constantVariable = "new value"; // TypeError
```

```
14
15 // const with objects and arrays
16 const person = { name: "John" };
17 person.name = "Jane"; // Allowed - modifying property
18 // person = { name: "Bob" }; // Error - reassigning
19
20 const numbers = [1, 2, 3];
21 numbers.push(4); // Allowed - modifying array
22 // numbers = [5, 6, 7]; // Error - reassigning
```

### Variable Scope

Listing 1.4: **Understanding Variable Scope**

```
1  // Global scope
2  var globalVar = "I'm global";
3  let globalLet = "I'm also global";
4
5  function demonstrateScope() {
6      // Function scope
7      var functionScoped = "Available throughout function";
8
9      if (true) {
10         // Block scope
11         var varInBlock = "Still function-scoped";
12         let letInBlock = "Only in this block";
13         const constInBlock = "Also only in this block";
14
15         console.log(functionScoped); // Works
16         console.log(varInBlock);     // Works
17         console.log(letInBlock);     // Works
18     }
19
20     console.log(functionScoped); // Works
21     console.log(varInBlock);     // Works (var ignores block)
22     // console.log(letInBlock);  // ReferenceError
23     // console.log(constInBlock); // ReferenceError
24 }
```

### Hoisting

Hoisting is JavaScript's default behavior of moving declarations to the top of their scope before code execution.

Listing 1.5: **Understanding Hoisting with var, let, and const**

```
// Hoisting with var
console.log(hoistedVar); // undefined (not error)
var hoistedVar = "I'm hoisted";
console.log(hoistedVar); // "I'm hoisted"

// The above is interpreted as:
// var hoistedVar; // Declaration hoisted
// console.log(hoistedVar); // undefined
// hoistedVar = "I'm hoisted"; // Assignment stays

// Hoisting with let/const (Temporal Dead Zone)
// console.log(notAccessible); // ReferenceError
let notAccessible = "I'm in the TDZ";

// Function hoisting
console.log(myFunction()); // "Works!"
function myFunction() {
    return "Works!";
}

// Function expressions are not hoisted
// console.log(myExpression()); // TypeError
var myExpression = function() {
    return "Not hoisted";
};

// Temporal Dead Zone (TDZ) demonstration
function demonstrateTDZ() {
    // TDZ starts for 'temp'
    // console.log(temp); // ReferenceError
    let temp = "Now accessible"; // TDZ ends
    console.log(temp); // "Now accessible"
}
```

## Data Types in JavaScript

JavaScript is dynamically typed with seven primitive types and one non-primitive type (Object).

### Primitive Data Types

Listing 1.6: **All Primitive Data Types**

```
// 1. Number - Integers and floating-point
let integer = 42;
let float = 3.14159;
let exponential = 2.5e3;  // 2500
let hexadecimal = 0xFF;   // 255
let binary = 0b1010;      // 10
let octal = 0o12;         // 10

// Special numeric values
let infinity = Infinity;
let notANumber = NaN;

// 2. String - Text data
let singleQuote = 'Hello';
let doubleQuote = "World";
let templateLiteral = `Hello ${doubleQuote}`;

// 3. Boolean
let isTrue = true;
let isFalse = false;

// 4. Undefined
let undefinedVar;
console.log(undefinedVar); // undefined

// 5. Null
let nullVar = null;

// 6. Symbol (ES6)
let sym1 = Symbol('id');
let sym2 = Symbol('id');
console.log(sym1 === sym2); // false

// 7. BigInt (ES2020)
let bigInteger = 123n;
let bigFromString = BigInt("9007199254740992");
```

### Type Conversion and Coercion

Listing 1.7: **Type Conversion and Coercion**

```
// Explicit Conversion
let num = 123;
let str = String(num);          // "123"
let strToNum = Number("456"); // 456
let bool = Boolean(1);          // true

// Implicit Coercion
console.log("5" + 3);     // "53" (string)
console.log("5" - 3);     // 2 (number)
console.log(true + 1);    // 2

// Falsy values
// false, 0, -0, 0n, "", null, undefined, NaN

// Equality comparisons
console.log(5 == "5");    // true (coercion)
console.log(5 === "5");   // false (strict)
```

## Arrays in JavaScript

Arrays are ordered collections that can hold multiple values of any type.

### Array Creation and Methods

Listing 1.8: **Arrays - Creation and Methods**

```
// Creating arrays
let numbers = [1, 2, 3, 4, 5];
let mixed = [1, "two", true, null];

// Accessing elements
console.log(numbers[0]);     // 1
console.log(numbers.length); // 5

// Mutating methods
numbers.push(6);          // Add to end
numbers.pop();            // Remove from end
numbers.unshift(0);       // Add to start
numbers.shift();          // Remove from start
numbers.splice(2, 1);     // Remove at index
numbers.sort();           // Sort in place
numbers.reverse();        // Reverse in place

// Non-mutating methods
```

```
let subset = numbers.slice(1, 3);
let joined = numbers.join("-");
let index = numbers.indexOf(3);

// Iteration methods (ES5+)
let doubled = numbers.map(n => n * 2);
let evens = numbers.filter(n => n % 2 === 0);
let sum = numbers.reduce((acc, n) => acc + n, 0);
let found = numbers.find(n => n > 3);

numbers.forEach((num, index) => {
    console.log(`Index ${index}: ${num}`);
});
```

## Strings in JavaScript

Strings represent text data and are immutable.

Listing 1.9: **String Creation and Methods**

```
// Creating strings
let single = 'Single quotes';
let double = "Double quotes";
let template = `Template literal with ${double}`;

// String methods
let text = "  JavaScript  ";
console.log(text.length);           // 14
console.log(text.trim());           // "JavaScript"
console.log(text.toUpperCase());    // "  JAVASCRIPT  "
console.log(text.toLowerCase());    // "  javascript  "

// Searching
let sentence = "The quick brown fox";
console.log(sentence.indexOf("quick"));     // 4
console.log(sentence.includes("fox"));      // true
console.log(sentence.startsWith("The"));    // true

// Extracting
console.log(sentence.substring(4, 9));  // "quick"
console.log(sentence.slice(-3));        // "fox"

// Modifying (returns new string)
console.log(sentence.replace("fox", "dog"));
console.log(sentence.split(" ")); // Array of words
```

## Functions in JavaScript

Functions are reusable blocks of code that perform specific tasks.

Listing 1.10: **Function Types and Features**

```
// Function Declaration (hoisted)
function greet(name) {
    return "Hello, " + name;
}

// Function Expression
const multiply = function(a, b) {
    return a * b;
};

// Arrow Function (ES6)
const square = x => x * x;
const sum = (a, b) => a + b;

// Default parameters (ES6)
function greetWithDefault(name = "World") {
    return `Hello, ${name}!`;
}

// Rest parameters (ES6)
function sumAll(...numbers) {
    return numbers.reduce((acc, n) => acc + n, 0);
}

// Destructuring parameters
function createUser({name, age, email = "N/A"}) {
    return { name, age, email, id: Date.now() };
}

// Closures
function createCounter(start = 0) {
    let count = start;
    return {
        increment: () => ++count,
        decrement: () => --count,
        getValue: () => count
    };
}

// Higher-order function
function withLogging(fn) {
```

```
    return function(...args) {
        console.log(`Calling with: ${args}`);
        return fn(...args);
    };
}

// IIFE (Immediately Invoked Function Expression)
(function() {
    console.log("IIFE executed!");
})();
```

## Objects and Methods

Objects are collections of key-value pairs.

Listing 1.11: **Objects - Creation and Manipulation**

```
// Object literal
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 30,

    // Method
    getFullName() {
        return `${this.firstName} ${this.lastName}`;
    }
};

// Accessing properties
console.log(person.firstName);       // Dot notation
console.log(person["lastName"]);     // Bracket notation
console.log(person.getFullName());   // Method call

// Adding/modifying properties
person.email = "john@example.com";
person.age = 31;

// Deleting properties
delete person.email;

// Object methods
console.log(Object.keys(person));    // Array of keys
console.log(Object.values(person)); // Array of values
console.log(Object.entries(person)); // Array of [key, value]

```

```
// Object destructuring
const { firstName, age } = person;

// Spread operator
const copied = { ...person };
const merged = { ...person, city: "NYC" };

// Constructor function
function Car(make, model, year) {
    this.make = make;
    this.model = model;
    this.year = year;
}

const myCar = new Car("Toyota", "Camry", 2022);

// ES6 Class
class Animal {
    constructor(name) {
        this.name = name;
    }

    speak() {
        return `${this.name} makes a sound`;
    }
}

const dog = new Animal("Buddy");
```

## Decisions and Loops

Control flow statements control the execution path of your program.

### Conditional Statements

Listing 1.12: **Decision Making Structures**

```
// if...else
let age = 18;
if (age < 18) {
    console.log("Minor");
} else if (age < 65) {
    console.log("Adult");
} else {
    console.log("Senior");
```

```
}

// switch
let day = 1;
switch (day) {
    case 0:
        console.log("Sunday");
        break;
    case 1:
        console.log("Monday");
        break;
    default:
        console.log("Other day");
}

// Ternary operator
let status = age >= 18 ? "adult" : "minor";

// Nullish coalescing (ES2020)
let username = null;
let displayName = username ?? "Guest";

// Optional chaining (ES2020)
let user = { profile: { name: "Alice" } };
console.log(user?.profile?.name); // "Alice"
console.log(user?.settings?.theme); // undefined
```

### Loops

Listing 1.13: **Loop Structures**

```
// for loop
for (let i = 0; i < 5; i++) {
    console.log(i);
}

// while loop
let count = 0;
while (count < 5) {
    console.log(count);
    count++;
}

// do...while loop
```

```
let num = 0;
do {
    console.log(num);
    num++;
} while (num < 3);

// for...in (objects)
const obj = { a: 1, b: 2, c: 3 };
for (let key in obj) {
    console.log(`${key}: ${obj[key]}`);
}

// for...of (iterables)
const arr = [1, 2, 3];
for (let value of arr) {
    console.log(value);
}

// break and continue
for (let i = 0; i < 10; i++) {
    if (i === 5) break;      // Exit loop
    if (i % 2 === 0) continue; // Skip iteration
    console.log(i);
}

// Array iteration methods
[1, 2, 3].forEach(n => console.log(n));
let doubled = [1, 2, 3].map(n => n * 2);
let evens = [1, 2, 3, 4].filter(n => n % 2 === 0);
let sum = [1, 2, 3].reduce((acc, n) => acc + n, 0);
```

# Chapter 2

# DOM MANIPULATION

*This chapter will cover Document Object Model manipulation...*

# Chapter 3

# FORMS AND REACT INTRODUCTION

*This chapter will cover HTML forms and introduction to React...*

# Chapter 4

# REACT STATE AND APIs

*This chapter will cover React state management and API integration...*

# Chapter 5

# MONGODB AND WEBPACK

*This chapter will cover MongoDB NoSQL database and Webpack bundling...*